# The Parallelization of SPIDER on Distributed-Memory Computers Using MPI

Chao Yang[a*], Pawel A. Penczek[b], ArDean Leith[c], Francisco J. Asturias[d], Esmond G. Ng[a], Robert M. Glaeser[e], Joachim Frank[f]

[a]Lawrence Berkeley National Laboratory, Computational Research Division, Berkeley, CA 94720, US

[b]The University of Texas – Houston Medical Center, Department of Biochemistry and Molecular Biology, Houston, TX 77030, US

[c]Wadsworth Center, New York State Department of Health, Albany, NY 12201, US

[d]The Scripps Research Institute, Department of Cell Biology, La Jolla, CA 92037, US

[e]The University of California, Department of Molecular & Cell Biology & Physical Biosciences Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, US

[f]Howard Hughes Medical Institute, HRI, Wadsworth Center, New York State Department of Health, Albany, NY 12201, US

[*]Corresponding author. Email address: CYang@lbl.gov. Fax: 510-486-5812

## Abstract

We describe the strategies and implementation details we employed to parallelize the SPIDER software package on distributed-memory parallel computers using the Message Passing Interface (MPI). The MPI-enabled SPIDER preserves the interactive command line and batch interface used in the sequential version of SPIDER, thus does not require users to modify their existing batch programs. We show the excellent performance of the MPI-enabled SPIDER when it is used to perform multi-reference alignment and 3-D reconstruction operations on a number of different computing platforms. We point out some performance issues when the MPI-enabled SPIDER is used for a complete 3-D projection matching refinement run, and propose several ways to further improve the parallel performance of SPIDER on distributed-memory machines.

*Keywords*: 3-D structure determination; Electron microscopy; Parallel computing; Message Passing Interface

## 1. Introduction

SPIDER (System for Processing Image Data from Electron microscopy and Related fields) (Frank *et al.*, 1996) is one of the most widely used software packages for carrying out single particle image reconstruction of three-dimensional (3-D) macromolecular assemblies from cryo-electron microscopy (Cryo-EM) image data (Frank, 2006). Due to the large volume of data and high computational complexity involved in such a reconstruction, enabling SPIDER to run in parallel on a multi-processor computer system has been an essential part of the SPIDER software development. The development of the shared-memory parallel (SMP) version of SPIDER began in 1992. The SMP version of SPIDER is implemented by manually inserting OpenMP (Chandra *et al.*, 2000) directives into the sequential code to achieve loop-level and task-level parallelism. Both fine-grain and coarse-grain parallelism have been utilized. To enhance efficiency, some parts of the code (for example, multi-reference 2-D alignment) were modified to provide alternative modes of parallelization depending on the amount and size of the input data. The parallel SPIDER code has been used successfully by structural biologists for many years. However, in the last few years, the number of large-scale high-performance, shared-memory, parallel machines, such as the SGI Origin and Cray T90, has been declining rapidly due to the high cost associated with manufacturing and maintaining this type of machines. Even when they are available, the single processor performance on these machines lags far behind the peak performance delivered by the latest Intel[1] and AMD[2] microprocessors. The current generation of shared-memory parallel systems are either extremely expensive (e.g., SGI Altix, Cray X1 and NEC SX-6), hence not widely accessible, or equipped with a small number of processors packed on a single board (node). On the other hand, clusters of PCs built on commodity Intel and AMD processors and running the Linux operating system have become widely available. They tend to be significantly less expensive than large-scale shared-memory machines. However, these machines do not share memory beyond a single node. Hence, the SMP version of SPIDER cannot be used directly on these machines to take advantage of the vast amount of processing and memory resources.

[1] http://developer.intel.com/products/processor/xeon/index.htm
[2] http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/30579_hi.pdf

To ameliorate this situation, we developed a parallel version of SPIDER that can be used on distributed-memory parallel computers such as the IBM SP, as well as clusters of PCs, by adopting the Message Passing Interface (MPI) (Pacheco, 1996) to distribute the data and perform necessary communications among different processors. The use of MPI allows us to reuse most of the sequential SPIDER code and make changes only at places that require data distribution and communication. Because MPI is supported by almost all parallel computer vendors, portability of the MPI-enabled SPIDER is automatically guaranteed. We made an effort to maintain the sequential SPIDER interactive command line and batch interfaces so that the current SPIDER users accustomed to the existing programming style would not need to change their SPIDER batch files in order to take advantage of the MPI-enabled SPIDER. Once the MPI-enabled SPIDER is installed on a parallel computer equipped with an MPI library, the user can employ the parallel software in the same way the sequential version of SPIDER is used.

Here we report the progress we have made in terms of the functionality and performance of the newly developed MPI version of SPIDER and provide test results that illustrate the tremendous advantages of processing electron microscope (EM) data in the MPI mode. We also discuss the difficulties we encountered in the development process and point to general directions that we believe should be taken in the design of the next generation of single particle software packages in order to ease the parallelization process while achieving the best performance on distributed-memory machines.

The paper is organized as follows. In Section 2, we will review the basic algorithmic ingredients of single-particle reconstruction implemented in SPIDER and the opportunities for their parallelization. In Section 3, we discuss implementation details and demonstrate the performance of MPI-enabled SPIDER on two major operations in a structure refinement procedure: alignment and 3-D reconstruction. In Section 4, we discuss a number of challenges encountered in the development of the MPI version of SPIDER and suggest several possible ways to improve the current implementation.

## 2. Parallelization Strategy

Before we discuss strategies for parallelizing a single-particle reconstruction task on a distributed-memory system, it is worthwhile to examine the mathematical formulation of the reconstruction problem and the algorithmic ingredients of the existing SPIDER software. For the purpose of this paper, we assume that isolated two-dimensional (2-D) single-particle images, each containing $n \times n$ pixels, have been selected from electron micrographs obtained experimentally, and the contrast transfer function (CTF) and envelope parameters associated with these images have been estimated. We are concerned only with using the collected 2-D images to compute the three-dimensional (3-D) density of the macromolecular assembly.

Formally, we state the estimation of the 3-D electron density map (denoted by $f \in R^{n^3}$) of a biological molecule from a large number of 2-D electron microscopy projection images, $b_i \in R^{n^2}$ ($i$=1,2,…$m$), of isolated (single) particles with random and unknown orientations as a nonlinear optimization problem:

$$\min_{\phi_i,\theta_i,\psi_i,x_i,y_i,f} \rho(\phi_i,\theta_i,\psi_i,x_i,y_i,f) \equiv \frac{1}{2}\sum_{i=1}^{m}\left\| T(x_i,y_i)P(\phi_i,\theta_i,\psi_i)f - b_i \right\|^2 , \qquad (1)$$

where $P(\phi_i,\theta_i,\psi_i)$ is a line integral operator that projects $f$ onto a 2-D plane after $f$ is rotated by a set of unknown Euler angles $\phi_i$, $\theta_i$, and $\psi_i$. Subsequently, the projection image is shifted by the translational operator $T(x_i, y_i)$. The factor of 1/2 is included merely for convenience. The objective function in (1) is clearly nonlinear due to the coupling between the orientation and translational parameters $\phi_i, \theta_i, \psi_i, x_i, y_i$ ($i$=1,2,…$m$) and the 3-D density $f$. The total number of unknown parameters to be computed is $n^3 + 5m$. Note that in single-particle analysis, the number of projection data $m$ is far greater than the linear size of the data in pixels, i.e., $m >> n$.

Currently, the optimization problem (1) is solved in two major phases. In the first phase, a low-resolution initial approximation of $f$ is obtained either experimentally, using the random conical tilt technique (Radermacher *et al.*, 1986), or computationally, using either the common-lines algorithms (Goncharov *et al.*, 1987; Penczek *et al.*, 1996; van Heel, 1987) or *ab initio* alignment of 2-D class averages (Ludtke *et al.*, 2004; Mullapudi *et al.*, 2004). Although some of these approaches can be computationally demanding, no consensus has been reached on the best choice among these methods, so we did not attempt to parallelize these methods. In the second phase, the 3-D structure *f* and the parameters $\phi_i, \theta_i, \psi_i, x_i, y_i$ ($i$=1,2,...$m$) are refined by a generalized coordinate descent scheme called 3-D *projection matching* (Penczek *et al.*, 1994). The method seeks a minimizer of (1) in two alternating search directions:

1. Starting from a given low-resolution density approximation $f_0$, the algorithm performs an exhaustive search for the optimal Euler angles $\phi_i, \theta_i, \psi_i$ and a restricted search for the optimal translations $x_i, y_i$ associated with each EM projection image $b_i$. These searches, which are implemented using 2-D *alignment* techniques (Joyeux and Penczek, 2002), are carried out by comparing $b_i$ with a set of reference projections $p_j$ ($j$=1,2,…,$m_r$) produced by computationally re-projecting $f_0$ in directions specified by a set of prescribed and quasi-uniformly distributed Euler angles $\hat{\phi}_j, \hat{\theta}_j, \hat{\psi}_j$ ($j$=1,2,…,$m_r$). The set of angles and shifts that yields a minimum value of $\|b_i - p_j\|$ (or, equivalently, a maximum cross-correlation coefficient) is assigned to $b_i$. The SPIDER commands for performing this type of orientation search are `AP SH` or `AP MQ`. This often constitutes the most time-consuming part of the refinement process. However, as we will see below, the exhaustive search of the orientation parameters can be easily parallelized.

2. Once each EM projection image has been assigned a set of updated Euler angles $\hat{\phi}_j, \hat{\theta}_j, \hat{\psi}_j$ and shifts $\hat{x}_j, \hat{y}_j$, a new density map $f_1$ is computed by solving a linear least-squares problem

$$\min_{f_1} \frac{1}{2} \sum_{j=1}^{m} \left\| T(\hat{x}_j, \hat{y}_j) P(\hat{\phi}_j, \hat{\theta}_j, \hat{\psi}_j) f_1 - b_j \right\|, \qquad (2)$$

preferably by using a version of the iterative algebraic reconstruction technique such as SIRT (SPIDER command `BP RP`), which yields a high-quality estimate of the density map (Penczek *et al.*, 2004). In what follows, we will refer to this step as 3-D reconstruction. After estimation of the resolution (using the Fourier Shell Correlation technique (Saxton and Baumeister, 1982)) and appropriate low-pass filtration of the current 3-D structure $f_1$ are completed, $f_1$ is used to begin the next cycle of the iterative process that results in a sequence of steadily improving – in terms of resolution, structures $f_1, f_2, \ldots, f_l$. The process continues until either changes of orientation parameters are deemed insignificant or the resolution does not improve any further.

The computational complexity for solving (2) is typically much lower than that associated with orientation search, although for large linear sizes of volumes it can be substantial. Unfortunately, the parallelization of the 3-D reconstruction step requires global communication to merge 2-D data into a 3-D volume. This can lead to some performance issues, as we will see below.

A flowchart for the projection matching algorithm is shown in Figure 1. The rectangular boxes in the figure represent data objects required or generated during the refinement process. The oval shaped boxes describe the operations performed on the data objects. The shaded boxes correspond to operations that we have parallelized using MPI. The implementation details of these operations will be described in the next section.

In order to reach high resolution, it is necessary to perform CTF correction (Penczek *et al.*, 1997) in the 3-D projection matching procedure. In SPIDER, this is done by dividing 2-D experimental images into groups of images with approximately the same defocus setting. The 2-D multi-reference alignment and 3-D reconstruction are carried out in each defocus group. The 3-D volumes produced in each defocus groups are merged together using the Wiener filtering technique (Frank *et al.*, 2000; Penczek *et al.*, 1997). Because our parallelization strategy does not exploit parallelism at the defocus level, we will not discuss CTF correction in this paper.

Assuming the set of reference projections is available, it is clear from the above problem formulation that the orientation search for one experimental image can be done independently from that of another. The first step of the 3-D projection matching method can be easily parallelized by dividing the experimental images into several groups and distributing them among different processors. If the set of reference images can be replicated on each processor, then the SPIDER multi-reference 2-D alignment can be executed simultaneously and independently on all processors. In this case, one would expect a nearly perfect parallel speedup if the cost of distributing the experimental images is relatively small.

The second step of the 3-D projection matching algorithm requires all experimental images to be ultimately merged within a single 3-D volume, regardless of the algorithm used to perform a 3-D reconstruction. When the reconstruction is carried out by the SIRT algorithm, the 3-D density of the macromolecule is updated iteratively as follows

$$f^{(k+1)} \leftarrow f^{(k)} + \lambda \sum_{i=1}^{m} P_i^T \left[\hat{b}_i - P_i f^{(k)}\right], \qquad (3)$$

where $k$ is the iteration number, $\lambda$ is a regularization parameter, and $P_i = P(\hat{\phi}_i, \hat{\theta}_i, \hat{\psi}_i)$ is a projection operator. Because the shifted 2-D images $\hat{b}_i$ ($i$=1,2,…$m$) are distributed on different processors during the alignment process, each processor can only perform a partial sum on the right hand side of (3). These partial sums must be collected and added together by a master processor before $f^{(k)}$ is updated. In our parallelization scheme, this updated volume is broadcast back to all processors before the next iteration of 3-D projection matching begins.

## 3. Implementation and Performance

In this section, we discuss the implementation of the parallel 3-D projection matching algorithm described in Section 2. We will also report the parallel performance of the algorithm for two different test problems on three different types of machines listed in Table 1. Table 1 also provides the single-processor peak performance characteristic of each system. The sustained single-processor performance of SPIDER on these machines depends on the combination of CPU speed, the number of instructions issued per clock cycle (which is reflected in the number of gigaflops per second), the cache and memory sizes, the memory bandwidth, the compiler, and the compiling options used (Table 2). For the purpose of this paper, we did not try to tune the SPIDER code or the compiling options to achieve the optimal single-processor performance because we are more interested in the parallel performance. The parallel scalability of the MPI-enabled SPIDER is affected by the performance of the network hardware, which we document in Table 3.

In addition to trying to achieve the goal of enabling SPIDER to run efficiently on a distributed-memory parallel computer, we also made an effort to ensure that the MPI version of SPIDER is easy to use. In particular, the parallel implementation of SPIDER commands strictly preserves the existing interactive SPIDER command line interface. As a result, users do not need to modify their SPIDER batch scripts in order to use the MPI version on a distributed-memory parallel machine. This feature frees the user from the usually demanding task of learning how to write a parallel code.

The interactive mode of SPIDER allows the user to type specific SPIDER commands represented by a few letters on the command line after a SPIDER session is launched. SPIDER would prompt the user to enter additional parameters and/or output filenames based on the command the user types in. In Figure 2, we show an example of a SPIDER interactive session in which a set of experimental images contained in a file named `expimg` are aligned against reference images contained the file `refprj`.

Instead of entering one command or one parameter at a time, a user can store a sequence of SPIDER commands along with the parameters required by each command in a file and pass it to SPIDER on the command line. In this case, SPIDER will interpret the commands (including do-loops) using a built-in command interpreter and parameters stored in the file and execute them (Frank *et al.*, 1996). This processing mode is usually referred to as the batch mode. Because SPIDER contains condition and branch statements such as `IF`, `THEN`, `GOTO` and other programming constructs such as expression evaluation and procedures, it can be viewed as a primitive programming language. Hence the file that contains a list of SPIDER commands is often referred to as a SPIDER batch script.

In the sequential version of SPIDER, a command-line parser is implemented within the top level SPIDER subroutines to interpret a list of user-entered commands (either through the command line interface or through a SPIDER batch file.) When the parser recognizes a legal SPIDER command, it usually transfers control to a specific subroutine to handle the desired task. The subroutine may continue to parse the next few lines to collect input parameters and output filenames. However, each subroutine accomplishes a well defined task organized in a hierarchical fashion.

The modular design of the SPIDER software makes our parallel implementation of the 3-D projection matching method relatively easy. Our implementation reuses most of the SPIDER computational kernels with slight modifications to accommodate data distribution and collective communication. We use the communication primitives provided in MPI to manage data distribution and task synchronization. The MPI application program interface (API) has become a widely adopted standard for distributed-memory parallel programs. It is supported by almost all parallel computer vendors. Several highly efficient public-domain implementations of the MPI library are also available, and can be easily installed on most Linux clusters. In this paper, we use LAM-MPI (Burns *et al.*, 1994) on the Xeon clusters, MVAPICH (Liu *et al.*, 2004) on the Opteron cluster and the IBM proprietary implementation of the MPI on the IBM SP Power5 systems.

Two datasets are used to measure the performance of the parallel alignment and reconstruction procedures. In Table 4 we give the image size, the number of experimental images and the number of reference images generated. The TFIID dataset (Andel *et al.*, 1999) is a relatively small dataset that does not have multiple defocus groups (Table 4). The RNAPII dataset described in (Craighead and Asturias, 2003) contains multiple defocus groups of larger images. However, for the purpose of measuring the performance of alignment and 3-D reconstruction, we chose the defocus group that contains the largest number of experimental images (Table 4). Because in the strategy tested with the MPI-enabled SPIDER one defocus group of images is processed at a time, the parallel performance of the code is determined by the performance of the alignment and 3-D reconstruction within each group. It is not affected by the number of defocus groups.

## *3.1. Parallel Alignment*

In our MPI implementation, we assume the number of processors (`ncpus`) used in the computation is fixed by the user. Each processor is assigned a distinct process identification number upon an MPI initialization call `MPI_INIT()`. We will use `mypid` to denote this identification number. Many SPIDER modules read images and volumes from the disk. In the MPI implementation, all the disk I/O operations are performed by a single processor.

When the amount of memory available on each processor is sufficient to hold the entire set of reference images, the parallel implementation of the SPIDER `AP SH` command reads all reference images from a single processor, which we will call the master. The master processor then broadcasts these images to all other processors through a call `MPI_BCAST()`. Details on the calling sequences of all MPI subroutines can be found in (Gropp and Snir, 1998).

The number of experimental images is typically very large (currently in the range of 10,000 to 500,000), so usually it is not possible to place all of them in the memory of a single processor.

Instead, in our MPI implementation of the 3-D projection matching method, these images are read into a buffer incrementally by the master processor and sent to other processors through a point-to-point communication call `MPI_SEND()`. All other processors use the `MPI_RECEIVE()` call to receive data from the master processor. An `IMG_LOC` array is allocated on each processor to hold a subset of images assigned to that processor. The master processor needs to allocate space for both the I/O buffer and the `IMG_LOC` array, and the images assigned to the master processor are copied into the `IMG_LOC` array after they are read into the buffer. For a homogeneous parallel computing system where all processors are of the same type and speed, a uniform partition of the experimental images is used. The pseudocode shown in Figure 3 illustrates how data distribution is carried out in our MPI implementation of the orientation search. Such a simple synchronized read and distribution scheme also allows the user to place image data on a local disk which sometimes has better I/O performance. It is possible to improve this scheme by eliminating the extra buffer space or by making use of MPI-IO functions. However, because data distribution currently does not represent a significant amount of communication overhead, we will not discuss advanced data distribution schemes here.

Once each processor has received its portion of the experimental images, very little modification is required to initiate the parallel alignment. Instead of passing all experimental images stored in the memory one by one to the alignment module, the MPI implementation passes the distributed images to the same module on each processor. As a result, each processor will execute the same alignment procedure, although on different images. In Figure 4 we provide a code snippet comparison between the sequential code and the MPI-enabled parallel code. Notice that in the MPI-enabled code, the loop index `IMI` varies from 1 to `NIMG_LOC`, where `NIMG_LOC` is the number of images that have been distributed to a local processor (different processors may have different `NIMG_LOC` values if `NIMG` is not divisible by `ncpus`). Also notice that the orientation parameters are returned in the local array `DLIST_LOC` instead of `DIST`.

In Figure 5, we show the wall-clock time consumed by the parallelized SPIDER multi-reference alignment on three different computing platforms listed in Table 1. The alignment was performed on the TFIID data set. We executed the alignment command using different numbers of processors to assess the parallel scalability of the alignment computation. It is easy to observe from Figure 5 that the MPI-enabled alignment procedure scales linearly with respect to the number of processors used on all three platforms. We should highlight the fact that on both the Opteron and the Power 5, executing the command in parallel by using the MPI-enabled SPIDER on 64 processors reduces the wall-clock time required in the alignment of the TFIID dataset from two hours to roughly two minutes representing a 60-fold speedup. On the Xeon cluster, which has a faster CPU clock but smaller cache and lower memory bandwidth, the speedup from four hours to 4 minutes is also linear. We should note that the wall-clock time reported in Figure 5 includes the time used to distribute experimental images, which is negligible in all cases. Similar performance is observed when the same test is preformed using the RNAPII dataset.

## *3.2. Parallel 3-D Reconstruction*

A number of 3-D reconstruction algorithms are implemented in SPIDER (Penczek *et al.*, 2004). Both the direct Fourier inversion method (`BP 3F`) and the iterative reconstruction methods such as SIRT (`BP RP`) and Conjugate Gradient (`BP CG`) have been parallelized using MPI. We will focus on the parallelization of the iterative reconstruction methods in this paper because they are often the recommended choice and they tend to be more time-consuming than the direct Fourier inversion.

In each iteration of SIRT or CG, we perform $m$ projection and back-projection operations, as indicated in Section 2, where $m$ is the number of experimental images used in the reconstruction. Because the orientation parameters returned from the parallel alignment procedure are distributed on different processors, the projection of an intermediate 3-D volume $\hat{f}$ in directions specified by the updated orientation parameters is automatically parallelized by performing line integrations of $\hat{f}$ along the projection directions specified by the distributed Euler angles. When the intermediate 3-D volume is replicated on each processor, no communication is required in the projection operation. However, the back-projections from distributed 2-D images must be added together within a single 3-D volume. The accumulated volume must then be broadcast to all processors before the next iteration can begin. By using MPI, the merge of the partial sum and the broadcast of the accumulated volume can be accomplished with a single MPI call, `MPI_ALLREDUCE()`. Most MPI libraries employs buffering techniques and tree-based algorithms to reduce the volume of data transfer in such an operation (Bernaschi and Iannello, 1998), so the use of the

`MPI_ALLREDUCE()` call provides optimal communication performance and simplifies the coding for such global communication.

In Figure 6 we show the performance of the parallel SIRT reconstruction algorithm (`BP RP`) on three different platforms when it is used to compute a 3-D reconstruction on the TFIID dataset. In this case, linear scalability is achieved on both the Opteron cluster and the IBM SP (Power5). The amount of communication required in the collective communication call `MPI_ALLREDUCE()` is negligible. This is due to the high bandwidth of the communication networks installed on these systems. On the Xeon cluster, which has a network with much lower bandwidth, the parallel `BP RP` command only scales linearly up to 8 processors. But there is still noticeable reduction in wall-clock time when the command is executed in parallel on 16 processors. Beyond that point, the cost of the collective communication increases significantly and the wall-clock time consumed by the 64-processor run exceeds those consumed by the 16- and 32-processor runs.

Analysis of Figure 6 reveals the potential communication bottleneck associated with the 3-D reconstruction step. If the parallel computation is performed on a dataset consisting of $m$ images using $p$ processors, the ratio of computation over the communication volume is $m/p$, which is independent of the image size. As we increase $p$ while holding $m$ constant, the computation/communication ratio will decrease. Such a reduction will ultimately lead to performance degradation in the parallel 3-D reconstruction even on hardware with very efficient network connection. This observation indicates that, depending on the hardware used and the size of the dataset processed, other data processing strategies might be considered if performance becomes an issue. Sometimes it may be more efficient to use a subset of the available processors in the 3-D reconstruction step to reduce the communication overhead. Such a decision can be made automatically at runtime based on the estimated ratio of computation over communication. When the estimated ratio is below a predefined threshold, a separate communication subgroup can be created by using the MPI function `MPI_GROUP_CREATE( )`, and the 3-D reconstruction can be computed within such a subgroup with improved efficiency. A user does not need to redistribute data manually.

## *3.3. Running a Complete Refinement Procedure in Parallel*

It is worth noting that the 3-D projection matching algorithm is only available in the form of a SPIDER batch program which contains a refinement loop that repeatedly invokes both the multi-reference 2-D alignment and 3-D reconstruction commands. In addition to these commands, the batch program also contains other basic input/output (I/O) and image manipulation commands. The parallelization of the multi-reference 2-D alignment and 3-D reconstruction procedures described in Sections 3.1 and 3.2 represents the first step in our effort to enable SPIDER to perform seamlessly and efficiently on a distributed-memory parallel-computer system. As the wall-clock time spent in these calculations is reduced through parallelization, other SPIDER operations that currently do not take a large percentage of overall time will become the major timing contributors in the performance profile. The parallelization of some of these operations will be relatively easy, while the parallel implementation of others may require major changes in SPIDER design or involve changing the way SPIDER batch programs are written by a user. We will discuss some of these issues in this section and propose some directions for future development in the next section. Before we begin, let us take a look at how the current MPI version of SPIDER performs on a complete refinement of the TFIID data set. We emphasize the fact that our parallel refinement does not require the user to make any changes to the existing SPIDER batch programs. Once MPI is installed and the MPI version of the SPIDER (called `spider_mpi`) is compiled, the user simply types, for example,

```
mpirun -np 16 spider_mpi prj @b98
```

in order to execute the batch program `b98` using dataset designated by the extension `prj` and the MPI version of SPIDER on 16 Opteron processors.

It is easy to observe from Figure 7 that the use of MPI-enabled SPIDER significantly reduces the runtime of the refinement when multiple processors are used on a cluster. However, the speedup in refinement is not linear, especially when the number of processors used exceeds 8.

A trivial reason why the parallel refinement procedure was not able to achieve linear scalability when a large number of processors are used is that the tested refinement procedure contains serial operations. In particular, the refinement strategy employed in this experiment separates the search for optimal Euler angles from the search for optimal translational parameters (shifts). The Euler angle search is parallelized; however, the translational alignment is implemented at the SPIDER batch script level. To be specific, the refinement batch file contains a SPIDER loop in which each image is rotated (with the SPIDER command `RT SQ`), cross-correlated with the selected reference image (SPIDER command `CC N`), and the peak of the cross-correlation function is found (SPIDER command `PK`) to yield the necessary translation. These simple image manipulation operations represent a small percentage of the total runtime when SPIDER is executed on a single processor. However, when the MPI-enabled SPIDER is executed on 64 processors, the high efficiency of the parallelized section makes these simple operations the most time-consuming part of the computation.

A less obvious reason that contributed to the sub-optimal parallel performance of the refinement procedure (when a large number of processors are used) is related to how I/O is implemented in the MPI-enabled SPIDER. The I/O operations involved in command-line interpretation are intrinsically sequential and do not need to be parallelized. Ideally, we can designate one processor to perform these operations and keep all other processors idle until a subroutine that contains parallel computation is invoked. However, such a scheme will amount to extensive changes in SPIDER source code because command line parsing operations are, to some extent, fused with the computational modules, especially those that are used to pass the user-defined parameters, do-loop indices etc.

To simplify the MPI implementation, we have taken an alternative approach in which user-entered commands are interpreted by all processors. That is, all processors follow the same execution path until a parallelized command is encountered. At that point, one of the processors may be designated to read and distribute image data while the other processors wait to receive image data. Once data distribution is completed, all processors resume simultaneous execution of the same set of instructions on different pieces of data. Under this scheme, non-parallelizable tasks that do not take much time are executed by all processors. Originally, we allowed all processors to read from the same file simultaneously while allowing only one processor to write to a file. Although this scheme works well most of the time, it can potentially fail when a read occurs right after a write. Due to I/O buffering and delays in some file systems, one processor may not immediately see a file created by a different node on the same network. This file coherence problem caused the previous version of MPI-enabled SPIDER to occasionally crash in an unpredictable manner. In the current implementation, we synchronize all I/O operations to allow only one processor to read and write. The data read by a single processor is broadcast to all other processors. This approach requires only a small number of changes at the lowest-level SPIDER I/O routines. The downside of this approach is that it introduces more communication overhead. Since each read is now followed by a broadcast, the latency cost increases significantly especially when a SPIDER non-image file that contains many parameters (such as angles) is read. In Figure 8 we give a snapshot of the communication pattern during the execution of a SPIDER procedure on 16 processors. The red blocks represent segments of time during which communication occurs. As we can see from this figure, communication occurs frequently throughout the time interval shown in the figure as a SPIDER command carries out some computation on data read from a non-image file line by line.

## 4. Discussion

We have used MPI to successfully parallelize the two most time-consuming SPIDER operations: the multi-reference alignment and 3-D reconstruction operations. These two operations jointly constitute the algorithmic core of the 3-D projection matching method used for refinement of EM structures in single particle reconstruction projects. We have demonstrated that the MPI-enabled multi-reference alignment and 3-D reconstruction SPIDER operations perform extremely well on a variety of distributed-memory parallel computer systems. In particular, the performance of the MPI implementation of these operations scales almost linearly with respect to the number of CPUs used on Linux clusters built with commodity processors and high speed interconnect.

The SPIDER interactive interface allows users to easily experiment with and combine different SPIDER commands to accomplish various image processing tasks. We generally tried to preserve this feature in the MPI version. However, it is not the most efficient way to carry out a large-scale EM structure refinements due to the overhead incurred in command line interpretation. Furthermore, the presence of condition and branch statements such as `IF` and `GOTO` makes the parallelization of SPIDER a difficult task. This type of problems are not limited to the SPIDER software package, but pertain to other interpretive language and programming environments such as MATLAB (Gilat, 2004) also. To further improve the parallel performance of the MPI version of SPIDER, a user may have to modify their SPIDER batch program to replace SPIDER batch loops and branches with simplified commands. However, such a modification should simplify the user's batch program.

Additional significant performance improvement of SPIDER system could be achieved by exploiting the do-loop level of parallelism that appears in a SPIDER batch script. Most of these do-loops are used to manipulate a stack of images. Conceptually, because the manipulation of one image is completely independent from another, these SPIDER loops should be parallelized by distributing images to different processors and performing the listed simple operations simultaneously on all processors. However, such type of loop parallelism is difficult to implement at the SPIDER batch file level because the command line parser is written in a high-level programming language (FORTRAN) and embedded in computational modules, so the parallelization has to be built into the system.

Our future work will also address issues of memory usage. In the current MPI implementation of the SPIDER multi-reference alignment procedures we assume each processor has sufficient amount of memory to hold the entire set of reference projection images. When the image size becomes large and when the angular separation between the projection images becomes small, the number of reference images becomes large. In effect, it may not be possible to replicate all reference images on all processors. The current solution is to save the reference images in a disk file and read one reference image at a time into the memory during the parallel alignment process. This approach introduces a significant amount of I/O overhead and thus slows down the alignment process. There are two alternatives which we will implement in the future. One solution is to generate the reference projections in parallel, so that they are distributed among different processors. These projection images will be passed around in a cyclic fashion during the parallel alignment process. Special care must be taken to avoid a dead lock in communication. Another simpler solution is to have all processors generate reference projections incrementally, so that only a subset of the reference projections is stored in memory at a given time during the alignment of each experimental image. Unfortunately, both approaches would require combining the computation of the reference projections with the alignment procedure in a single command. This would force users to modify their existing SPIDER batch programs. However, an advantage is that such a change should simplify their batch programs.

The issue of memory limitation also arises in the 3-D reconstruction procedure. The iterative reconstruction algorithms implemented in SPIDER require storing three real-type volumes on each processor. The direct inversion reconstruction algorithms require padding volumes with zero to double the size which result in eight-fold increase of the memory requirement (Penczek *et al.*, 2004). As the image size increases, these volumes must be distributed among different processors. Such a change in distribution would require additional communication.

The MPI version of SPIDER can be used on a heterogeneous cluster when an MPI library capable of handling different flavors of portable operating system interface (POSIX) and data format (such as the LAM/MPI (Burns *et al.*, 1994)) is installed. However, because in the current data distribution scheme implemented in the MPI version of the SPIDER we assume all processors run at the same speed, using the MPI version of SPIDER on a cluster of processors with vastly different performance is generally not efficient. In such a computing environment, a finer-grained client-server parallel computing model, implemented as the SPIDER `PubSub` system, may be more appropriate.

## 5. Availability of the Software

The MPI version of SPIDER is included in the current SPIDER release that can be downloaded from http://www.wadsworth.org/spider_doc/spider/docs/spider.html. There is no separate source for the MPI implementation; all MPI-related codes are inserted using the `#ifdef` preprocessor macros. To generate an MPI version of the executable, one can simply modify the Makefile for creating the sequential executable and insert `-DUSE_MPI` in the compiler option. Sample Makefiles are provided in the SPIDER source directory.

## 6. Acknowledgments

This research has been supported by NIH grant P01 GM064692 (to RMG, JF, EGN and PAP). We would like to thank William Young for providing access to the Xeon Linux cluster at the Scripps Research Institute. We are grateful to Bimal Rath who helped with testing MPI-SPIDER at the Wadsworth Center. We would also like to thank Eva Nogales for providing the TFIID data set and

Slaton Lipscomb for creating and maintaining a very useful website (http://cryoem.berkeley.edu/~slaton/spider/index.shtml) on how to compile and build the MPI-version of SPIDER. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy.

## References

Andel, F., Ladurner, A. G., Inouye, C., Tjian, R., Nogales, E., 1999. Three-dimensional structure of the human TFIID-IIA-IIB complex. Science 286, 2153-2156.

Bernaschi, M., Iannello, G., 1998. Collective communication operations: experimental results vs. theory. Concurrency-Practice and Experience 10, 359-386.

Burns, G., Daoud, R., Vaigl, J., 1994. LAM: An Open Cluster Environment for MPI. Proceeding of Supercomputing Sumposium, 379-386.

Chandra, R., Kohr, D., Maydan, D., Dagum, L., McDonald, J., Menom, R., 2000. Parallel Programming in OpenMP. Academic Press, Boston.

Craighead, J. L., Asturias, F. J., 2003. Structural analysis of the RSC chromatin-remodeling complex. Proc. Natl. Acad. Sci. USA 100, 6893-5.

Frank, J., 2006. Three-Dimensional Electron Microscopy of Macromolecular Assemblies. Oxford University Press, New York.

Frank, J., Penczek, P., Agrawal, R. K., Grassucci, R. A., Heagle, A. B., 2000. Three-dimensional cryoelectron microscopy of ribosomes. Methods Enzymol. 317, 276-291.

Frank, J., Radermacher, M., Penczek, P., Zhu, J., Li, Y., Ladjadj, M., Leith, A., 1996. SPIDER and WEB: processing and visualization of images in 3D electron microscopy and related fields. J. Struct. Biol. 116, 190-199.

Gilat, A., 2004. MATLAB: An introduction with applications. Wiley, Hoboken, NJ.

Goncharov, A. B., Vainshtein, B. K., Ryskin, A. I., Vagin, A. A., 1987. Three-dimensional reconstruction of arbitrarily oriented identical particles from their electron photomicrographs. Sov. Phys. Crystallography 32, 504-509.

Gropp, W., Snir, M., 1998. MPI the Complete Reference. MIT Press, Cambridge.

Joyeux, L., Penczek, P. A., 2002. Efficiency of 2D alignment methods. Ultramicroscopy 92, 33-46.

Liu, J. X., Chandrasekaran, B., Yu, W. K., Wu, J. S., Buntinas, D., Kini, S., Panda, D. K., Wyckoff, P., 2004. Microbenchmark performance comparison of high-speed cluster interconnects. Ieee Micro 24, 42-51.

Ludtke, S. J., Chen, D. H., Song, J. L., Chuang, D. T., Chiu, W., 2004. Seeing GroEL at 6 Ångstrom resolution by single particle electron cryomicroscopy. Structure 12, 1129-1136.

Mullapudi, S., Pullan, L., Bishop, O. T., Khalil, H., Stoops, J. K., Beckmann, R., Kloetzel, P. M., Kruger, E., Penczek, P. A., 2004. Rearrangement of the 16S precursor subunits is essential for the formation of the active 20S proteasome. Biophys. J. 87, 4098-4105.

Pacheco, P. S., 1996. Parallel Programming with MPI. Morgan Kaufmann, San Francisco.

Penczek, P. A., Grassucci, R. A., Frank, J., 1994. The ribosome at improved resolution: new techniques for merging and orientation refinement in 3D cryo-electron microscopy of biological particles. Ultramicroscopy 53, 251-270.

Penczek, P. A., Renka, R., Schomberg, H., 2004. Gridding-based direct Fourier inversion of the three-dimensional ray transform. J. Opt. Soc. Am. A 21, 499-509.

Penczek, P. A., Zhu, J., Frank, J., 1996. A common-lines based method for determining orientations for N > 3 particle projections simultaneously. Ultramicroscopy 63, 205-218.

Penczek, P. A., Zhu, J., Schröder, R., Frank, J., 1997. Three-dimensional reconstruction with contrast transfer function compensation from defocus series. Scanning Microscopy Supplement 11, 1-10.

Radermacher, M., Wagenknecht, T., Verschoor, A., Frank, J., 1986. A new 3-D reconstruction scheme applied to the 50S ribosomal subunit of *E. coli*. J. Microsc. 141, RP1-RP2.

Saxton, W. O., Baumeister, W., 1982. The correlation averaging of a regularly arranged bacterial envelope protein. J. Microsc. 127, 127-138.

van Heel, M., 1987. Angular reconstitution: *a posteriori* assignment of projection directions for 3D reconstruction. Ultramicroscopy 21, 111-124.

## Tables

Table 1. The single node hardware performance of the parallel computer systems used in the experiment.

| machine | clock speed | Gflops | CPUs/node | cache | memory |
|---|---|---|---|---|---|
| Intel Xeon-EMT | 3.4 Ghz | 6.8 | 2 | 512KB | 4GB |
| AMD Opteron | 2.2 Ghz | 4.4 | 2 | 1MB | 4GB |
| IBM Power5 | 1.9 Ghz | 7.6 | 8 | 32MB | 32GB |

Table 2. Compilers and compiling options used on different computers.

| machine | compiler | options |
|---|---|---|
| Intel Xeon-EMT | ifort (Intel) | `-arch pn4 -O3 -WB -mp -fpp2 -auto` `-pc64 -w95 -tpp7 -xW -assume byterecl` |
| AMD Opteron | pathf90 (Pathscale) | `-march=opteron -O3 -byteswapio` |
| IBM Power5 | xlf90 (IBM) | `-O3 -qarch=pwr5 -qstrict -qtune=pwr5 -qfixed=72 -qnosave` |

Table 3. The network hardware performance on different computers used in our experiments.

| machine | connection | bandwidth | latency |
|---|---|---|---|
| Intel Xeon-EMT | gigabit ethernet | 125MB/s | $40\mu s$ |
| AMD Opteron | InfiniBand | 620MB/s | $4.5\mu s$ |
| IBM Power5 | Federation HPS | 2GB/s | $5.0\mu s$ |

Table 4. The number and size of images associated with the data sets used in the computational experiments.

| name | image size | number of experimental images | number of reference images |
|---|---|---|---|
| TFIID | 64 | 4,418 | 799 |
| RNAPII | 128 | 2,887 | 5,088 |

**Figure Captions**

Figure 1. A flowchart of the 3-D projection matching algorithm. The rectangular boxes labeled by 'reference projections', 'Euler angles and shifts', 'experimental images', as well as the cylinder shaped object labeled by '3-D model' represent either the input data or intermediate data produced by operations described in oval shaped boxes (such as 'project', 'align' and '3-D reconstruction'). The shaded oval boxes correspond to operations that have been parallelized in the MPI version of SPIDER.

Figure 2. An interactive SPIDER session in which the SPIDER 2-D multi-reference alignment command AP SH is used to align a set of experimental images contained in the file expimg using

images contained in the file refprj as the reference. Various parameters are passed through the interactive command line interface shown here. The output of the alignment is return in the file named apmq002.

Figure 3. A piece of pseudocode that shows how data distribution is achieved in the MPI version of the SPIDER. Here we assume the master processor which has the identification number (mypid = 0) read experimental images incrementally into a buffer and send them to different processors while the other processors whose identification numbers are greater than 0 receive a subset of the experimental images from the master processor.

Figure 4. A code snippet that shows the parallelization of the 2-D multi-reference alignment procedure in SPIDER using MPI requires only a slight modification in the arguments passed into the computational module APRQ2D.

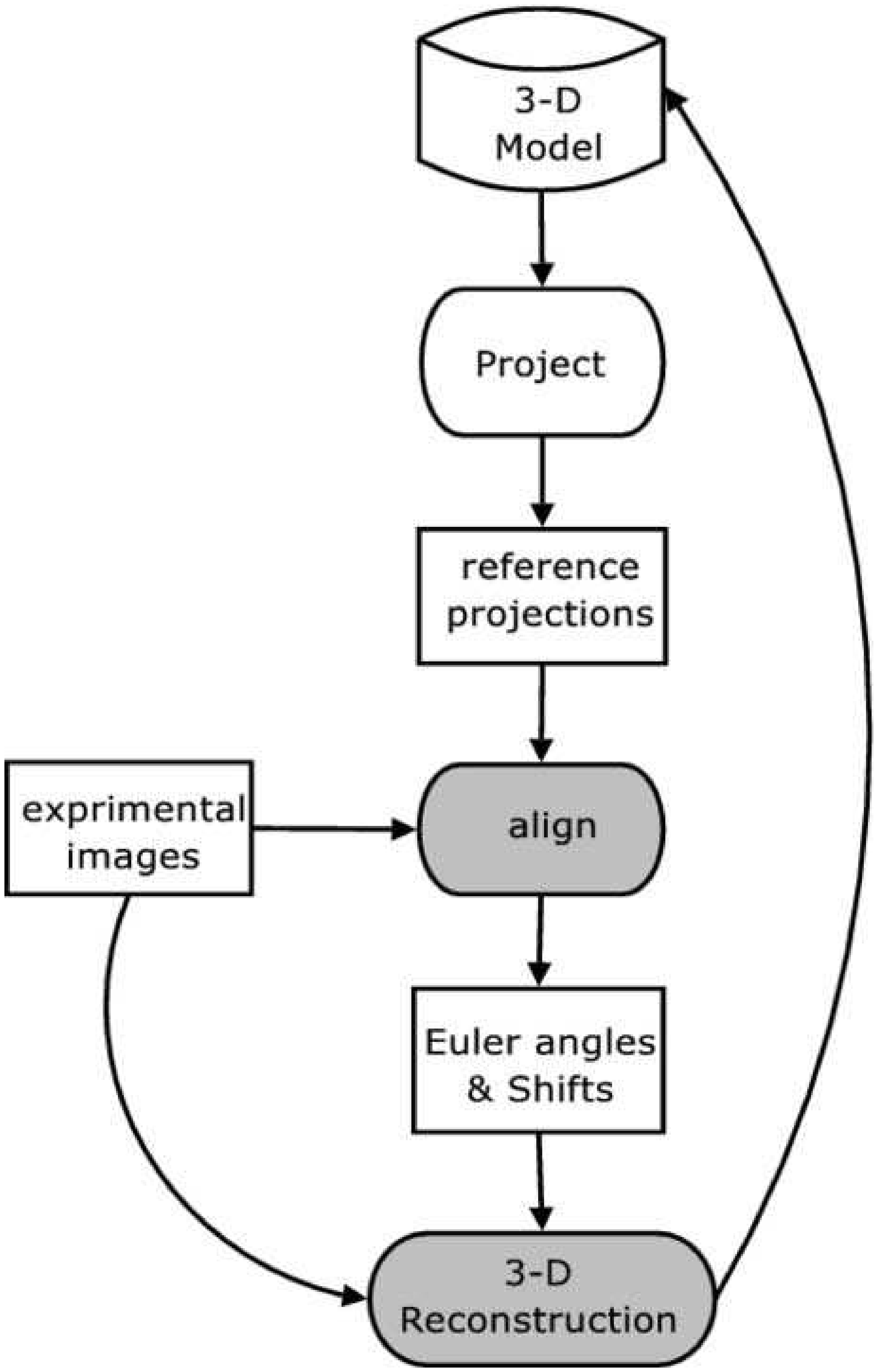
Figure 5. The parallel performance of the SPIDER AP SH command measured in wall-clock time on three different platforms when it is used on the TFIID dataset.

Figure 6. The parallel performance of the SPIDER BPRP 3-D command (which implements the SIRT 3-D reconstruction method) measured in wall-clock time on three different platforms.

Figure 7. The parallel performance of a complete SPIDER 3-D projection matching refinement applied to the TFIID dataset.

Figure 8. The communication pattern within an MPI-enable SPIDER run. The figure shows a small timeline display window. Time increases from left to right. Each horizontal green bar represents a processor. The thin lines extending between the bars represent the individual messages. Time spent in an MPI call is shown in red, and time spent in SPIDER computation is green.

Figure 1


3-D
Model
Project
reference
projections
exprimental
images
align
Euler angles
& Shifts
3-D
Reconstruction

**Figures 2-4**

```
.OPERATION: AP SH
.ENTER TEMPLATE FOR REFERENCE IMAGES: refprj@*****
.ENTER FILE NUMBERS OR SELECTION DOC.  FILE NAME: 1-799
.TRANSLATION SEARCH RANGE, STEP SIZE: 4 1
.FIRST LAST RING: 8 40
.REFERENCE IMAGES ANGLES DOCUMENT FILE:final/defgrp001/angvoea
.ENTER TEMPLATE FOR IMAGE SERIES TO BE ALIGNED: expimg@*****
.ENTER FILE NUMBERS OR SELECTION DOC.  FILE NAME: 1-2000
.EXPERIMENTAL IMAGES ALIGNMENT DOCUMENT FILE: *
.RANGE OF PROJECTION ANGLE SEARCH ANGLE CHANGE THRESHOLD: 0.00
.CHECK MIRRORED POSITIONS (0=NOCHECK / 1=CHECK)?: 1
.OUTPUT ALIGNMENT DOCUMENT FILE: final/defgrp001/apmq002
```

```
if (mypid = 0)
   for i = 0, 1, …, ncpus-1
     1. determine the global indices (ibeg and iend) of the
        first and last images that will be assigned to
        processor i;
     2. for j = ibeg,…, iend
              read the jth experimental images into buffer;
        endfor
     3. if (i>0)
           send data in the buffer to processor i;
        else
           copy data in the buffer to the IMG_LOC array;
        endif
   endfor
else
   receive data from mypid=0 into the IMG_LOC array;
endif
```

**Before:**

```
DO IMI = I, NIMG
   CALL APRQ2-D(IMG(1,1,IMI),...,DLIST(3,IMI),DLIST(4,IMI),...)
ENDDO
```

**After:**

```
DO IMI = I, NIMG_LOC
   CALL APRQ2-D(IMG_LOC(1,1,IMI),...,DLIST_LOC(3,IMI),
               DLIST_LOC(4,IMI),...)
ENDDO
```

**Figure 5**
**Click here to download high resolution image**

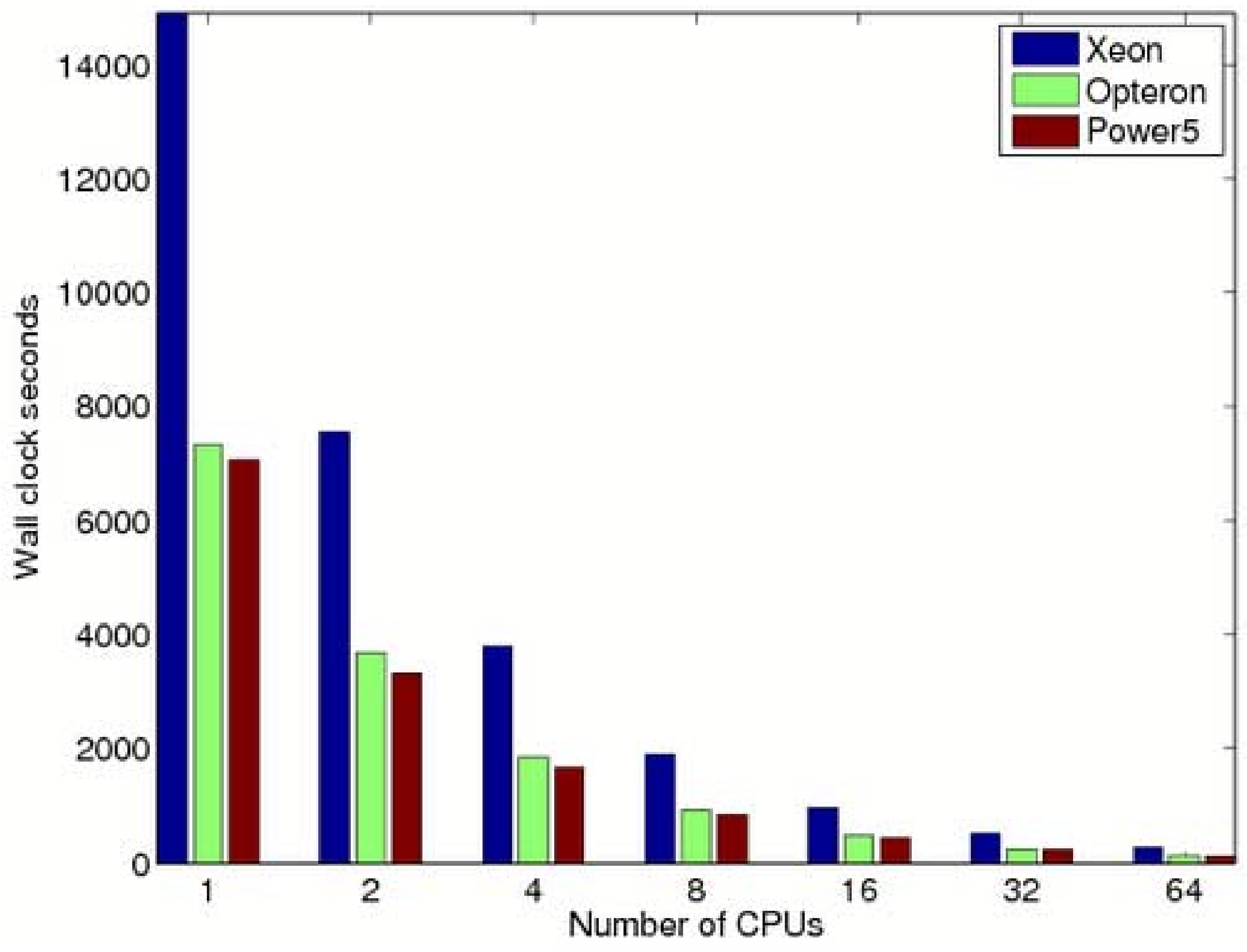
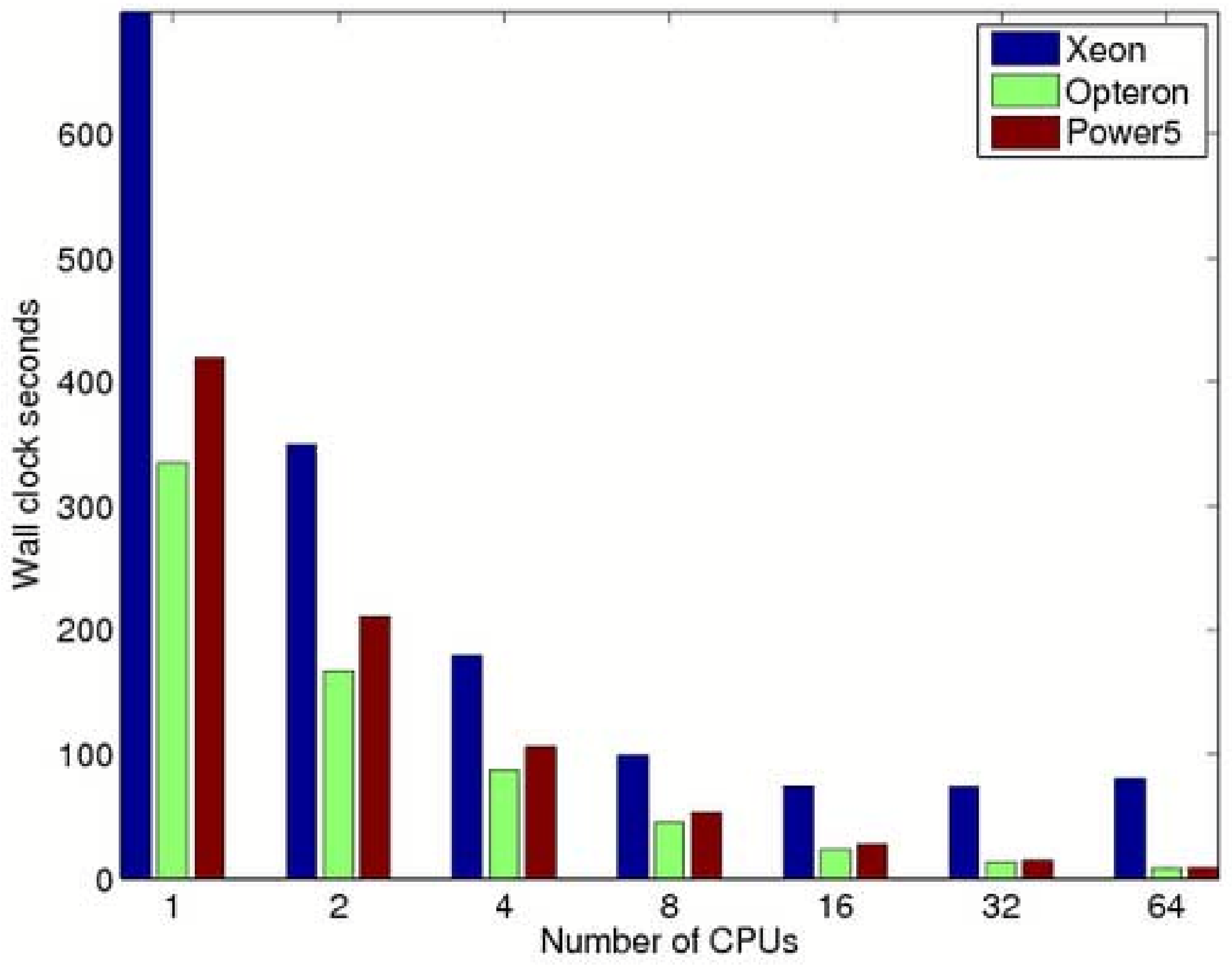


**Figure 6**
**Click here to download high resolution image**



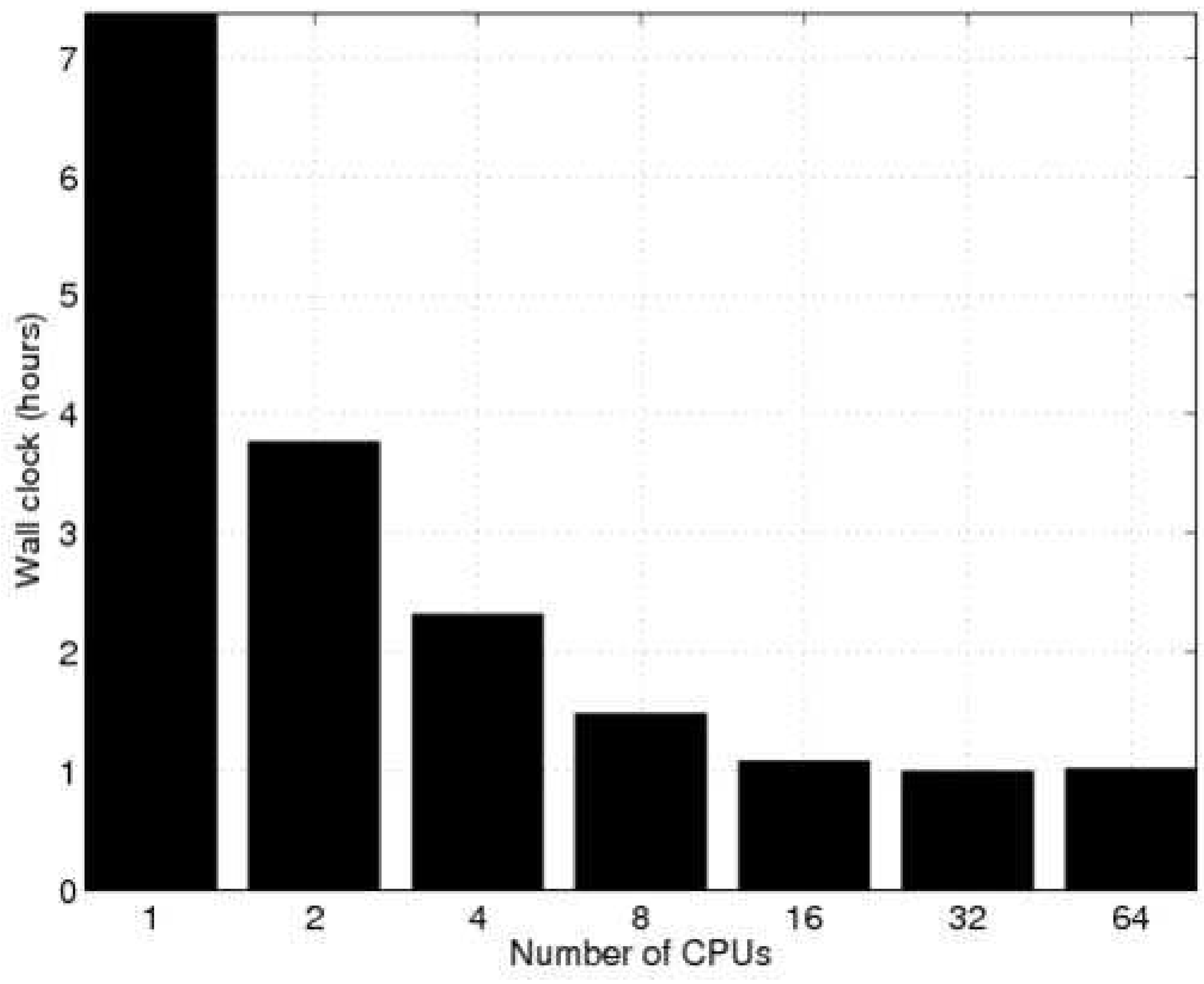
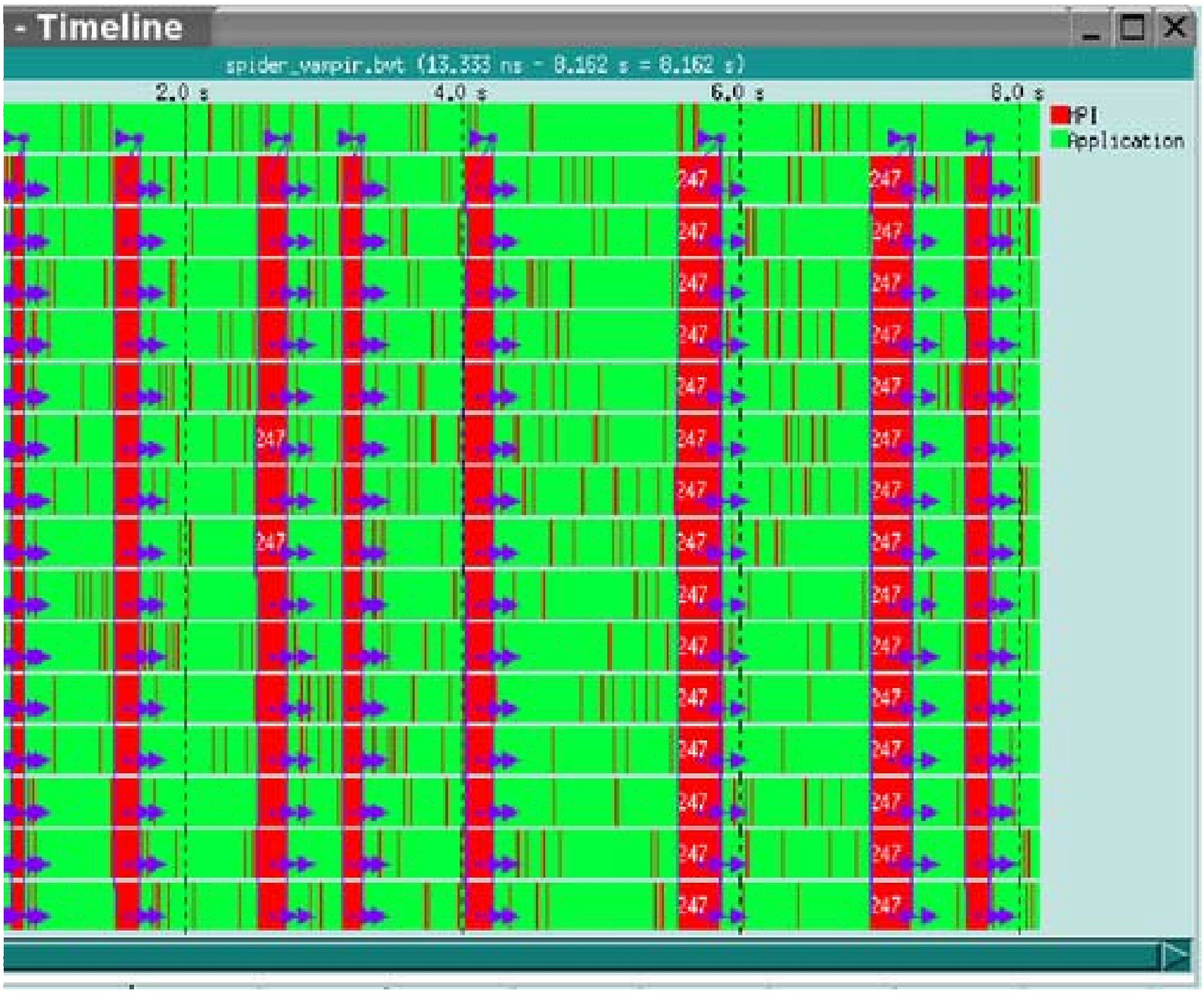

Wall clock (hours)
0
1
2
3
4
5
6
7
1
2
4
8
16
32
64
Number of CPUs

**Figure 8**
**Click here to download high resolution image**


- Timeline
spider_vampir.bvt (13.333 ns - 8.162 s = 8.162 s)
2.0 s
4.0 s
6.0 s
8.0 s
MPI
Application
247